OBJECT-ORIENTED PROGRAMMING
LAB 3: STRINGBUFFER, STRINGBUILDER, STRINGTOKENIZER

I. Objective

After completing this tutorial, you can:

- Understand how to program with **StringBuffer**, **StringBuilder**, and **StringTokenizer**.

II. The class *StringBuffer*

In some situations, it is useful to be able to alter the sequence of characters stored in a string. But class **String** supports only nonmutable string. To create mutable string use the class **StringBuffer** from the package `java.lang`. This class provides the same functionality as the class String, plus the following methods that actually modify the value stored in the **StringBuffer** object.

StringBuffer Constructors

1. **StringBuffer()**

Creates a **StringBuffer** with empty content and 16 reserved characters by default.

```
StringBuffer sb = new StringBuffer();
```

2. **StringBuffer(int sizeOfBuffer)**

Creates a **StringBuffer** with the passed argument as the size of the empty buffer.

```
StringBuffer sb = new StringBuffer(20);
```

3. **StringBuffer(String string)**

Creates a **StringBuffer** with the passed String as the initial content of the buffer. 16 contingent memory characters are pre-allocated, not including the buffer, for modification purposes.

```
StringBuffer sb = new StringBuffer("Hello World!");
```

StringBuffer Methods

1. **length()**

Returns the **StringBuffer** object's length.

```
StringBuffer sb = new StringBuffer("Hello");
int sbLength = sb.length();
System.out.println("String Length of " + sb + " is " + sbLength);
// String Length of Hello is 5
```

2. capacity()

Returns the capacity of the **StringBuffer** object.

```
StringBuffer sb = new StringBuffer("Hello");
int sbCapacity = sb.capacity();
System.out.println("Capacity of " + sb + " is " + sbCapacity);
// Capacity of Hello is 21
```

3. append()

Appends the specified argument string representation at the end of the existing String Buffer. This method is overloaded for all the primitive data types and Object.

```
StringBuffer sb = new StringBuffer("Happy ");
sb.append("New Year ");
sb.append(2020);
System.out.println(sb);
// Happy New Year 2020
```

4. insert()

The string str is inserted into this string buffer at the index indicated by offset. Any characters originally above that position are moved up and the length of this string buffer increased by the length of `str`. If `str` is null, the string null is inserted into this string buffer.

```
StringBuffer sb = new StringBuffer("HelloWorld ");
sb.insert(5, " ");
sb.insert(sb.length(), 2020);
System.out.println(sb);
//Hello World 2020
```

5. reverse()

Reverses the existing String or character sequence content in the buffer and returns it. The object must have an existing content or else a `NullPointerException` is thrown.

```
StringBuffer sb = new StringBuffer("Hello World");
System.out.println(sb.reverse());
//dlroW olleH
```

6. delete(int start, int end)

Removes the characters in a substring of this string buffer starting at index `start` and extending to the character at index `end - 1` or to the end of the string buffer if no such character exists. If `start` is equal to `end`, no changes are made. This method may throw `StringIndexOutOfBoundsException` if the value of `start` is negative, greater than the length of the string buffer, or greater than `end`.

```
StringBuffer sb = new StringBuffer("Hello World");
System.out.println(sb.delete(5,11));
//Hello
```

7. **setCharAt(int index, char ch)**

The character at index `indexof` this string buffer is set to `ch`. This method may throw `IndexOutOfBoundsException` if the value of index is negative or is greater than or equal to the length of the string buffer.

```
StringBuffer sb = new StringBuffer("Hello World");
System.out.println(sb.setCharAt(2, 'E'));
//HEllo World
```

8. **replace(int start, int end, String str)**

Replaces the characters in a substring of this string buffer with characters in the specified string `str`. The substring to be replaced begins at index `start` and extends to the character at index `end - 1` or to the end of the string buffer if no such character exists. The substring is removed from the string buffer, and then the string `str` is inserted at index `start`. If necessary, the string buffer is lengthened to accommodate the string `str`. This method may throw `StringIndexOutBoundsException` if the value of start is negative, greater than the length of the string buffer, or greater than `end`.

```
StringBuffer sb = new StringBuffer("Hello World!");
System.out.println(sb.replace(6,11,"Earth"));
//Hello Earth!
```

III. The class StringBuilder

Java **StringBuilder** class is mutable sequence of characters. **StringBuilder** Class can be comparable to String however the **StringBuilder** class provides more versatility because of its modification features.

- **StringBuilder** class provides an API similar to **StringBuffer**, but unlike **StringBuffer**, it doesn’t guarantee thread safety.
- Java **StringBuilder** class is designed for use as a drop-in replacement for **StringBuffer** in places where the string buffer was being used by a single thread (as is generally the case).
- If execution speed and performance is a factor, **StringBuilder** class can be used in place of **StringBuffer**.
- The bread-and-butter operations provided by the **StringBuilder** Class are the `append()` and `insert()` methods. These methods are overloaded within **StringBuilder** in order to accommodate different data type.

- The general process flow of **StringBuilder** append and insert methods is: (1) converts a given data to a string then (2) appends or inserts the characters of that string to the string builder. Java **StringBuilder** `append()` method always adds these characters at the end of the builder; `insert()` method inserts character(s) at a specified point.

StringBuffer and StringBuilder

StringBuffer	**StringBuilder**
Synchronized, hence thread safe.	Not synchronized, not thread safe.
Operates slower due to thread safety feature	Better performance compared to StringBuffer
Has some extra methods – substring, length, capacity etc.	Not needed because these methods are present in String too

StringBuilder Constructors

1. **StringBuilder()**

Creates an empty string builder with a default capacity of 16 (16 empty elements).

```
StringBuilder str = new StringBuilder();
```

2. **StringBuilder(CharSequence seq)**

Constructs a string builder containing the same characters as the specified CharSequence, plus an extra 16 empty elements trailing the CharSequence.

```
StringBuilder str1 = new StringBuilder("AAAABBBCCCC");
```

3. **StringBuilder(int capacity)**

Creates an empty string builder with the specified initial capacity.

```
StringBuilder str2 = new StringBuilder(10);
```

4. **StringBuilder(String str)**

Creates a string builder whose value is initialized by the specified string, plus an extra 16 empty elements trailing the string.

```
StringBuilder str3 = new StringBuilder("Hello World");
```

StringBuilder Methods

1. **append()**

StringBuilder `append()` method concatenates or attaches the passed String argument with the existing declared string. It attaches it after the declared string.

```
StringBuilder sb = new StringBuilder("Hello ");
sb.append("World");
System.out.println(sb);
// Hello World
```

2. **insert()**

StringBuilder `insert()` method inserts the passed String argument at the passed **String** index.

```
StringBuilder sb = new StringBuilder("HellWorld");
sb.insert(4, "o ");
System.out.println(sb);
// Hello World
```

3. **replace(int startIndex, int endIndex, String str)**

StringBuilder `replace()` method replaces the existing declared string. String replacement occurs from the passed `startingIndex` up to the `endingIndex`.

```
StringBuilder sb = new StringBuilder("Hello World!");
sb.replace(6,11,"Earth");
System.out.println(sb);
// Hello Earth!
```

4. **delete(int startIndex, int endIndex)**

StringBuilder `delete()` method deletes a character or sets of characters. Deletion occurs at passed `startingIndex` up to `endingIndex`.

```
StringBuilder sb = new StringBuilder("Dung Cam Quang");
sb.delete(0,9);
System.out.println(sb);
//Quang
```

5. **reverse()**

The `reverse()` method of **StringBuilder** class reverses the existing declared string. Invoking a `reverse()` method on a **StringBuilder** object with no existing declared value throws `NullPointerException`.

```
StringBuilder sb = new StringBuilder("devil");
sb.reverse();
System.out.println(sb);
// lived
```

6. capacity()

The `capacity()` method of **StringBuilder** class returns the current capacity of the **StringBuilder** object. The default capacity of the Builder is 16. If the number of character increases from its current capacity, it increases the capacity by (old_capacity2)+2 e.g. at current capacity 16, it becomes (162)+2=34.

```
StringBuilder sb=new StringBuilder();
System.out.println(sb.capacity()); // default value 16
sb.append("Java");
System.out.println(sb.capacity()); // still 16
sb.append("Hello StringBuilder Class!");
System.out.println(sb.capacity()); // (16*2)+2
```

IV. The class StringTokenizer

Another useful class when working with strings is **StringTokenizer** in the package `java.util`. This class allows a program to break a string into pieces or tokens. The tokens are separated by characters known as delimiters. When you create a **StringTokenizer** instance, you must specify the string to be tokenized. Other constructors within **StringTokenizer** allow you to specify the delimiting characters and whether the delimiting characters themselves should be returned as tokens.

String Tokenizer Constructors

1. **StringTokenizer(String str)**

This constructor creates a string tokenizer for the specified string `str`. The tokenizer uses the default delimiter set, which is the space character, the tab character, the newline character, the carriage-return character, and the form feed character. Delimiter characters themselves are not treated as tokens.

```
StringTokenizer st1 = new StringTokenizer("Hello guy How are you");
```

2. **StringTokenizer(String str, String delim)**

This constructor creates a string tokenizer for the specified string `str`. All characters in the `delim` string are the delimiters for separating tokens. Delimiter characters themselves are not treated as tokens.

```
StringTokenizer st2 = new StringTokenizer("JAVA : Code : String", " :");
```

3. **StringTokenizer(String str, String delim, boolean returnTokens)**

This constructor creates a string tokenizer for the specified string `str`. All characters in the `delim` string are the delimiters for separating tokens. If the `returnTokens` flag is true, the delimiter characters are also returned as tokens. Each delimiter is returned as a string of length 1. If the flag is false, the delimiter characters are skipped and serve only as separators between tokens.

```
StringTokenizer st3 = new StringTokenizer("JAVA : Code : String", " :",  true);
```

StringTokenizer Methods

1. **nextToken()**

Returns the next token in the string. If there are no more tokens in the string, it throws the exception `NoSuchElementException`.

2. **hasMoreTokens()**

Returns true if the string contains more tokens.

```
StringTokenizer st1 = new StringTokenizer("Hello guy How are you", " ");
while (st1.hasMoreTokens())
{
    System.out.println(st1.nextToken());
}
// Hello
// guy
// How
// are
// you
```

3. **countTokens()**

Calculates the number of times that this tokenizer's nextToken method can be called before it generates an exception. The current position is not advanced. This function returns the number of tokens remaining in the string using the current delimiter set.

IV. Exercises

1. Write a Java program:
   - Split the full name into first name and last name (using StringTokenizer).
2. Write a Java program:
   - Count number of words in string (using StringTokenizer)
   - Concatenate one string contents to another (using StringBuffer or StringBuilder).
   - Check a string is palindrome or not (using StringBuffer or StringBuilder).
3. Write a Java program:
   - Remove leading white spaces from a string (using StringTokenizer and StringBuffer/StringBuilder).
   - Remove trailing white spaces from a string (using StringTokenizer and StringBuffer/StringBuilder).
   - Remove extra spaces from a string (using StringTokenizer and StringBuffer/StringBuilder).
4. Write a Java program:
   - Write a function that processes a String paragraph, count occurrences of each word in the paragraph, and store them in a 2D-array.
   - Write a main function, use the above function and print the matrix to the screen.

   For example: "I live in Ho Chi Minh city. I study at TDTU"

   Screen:

```
I 2
live 1
in 1
Ho 1
Chi 1
Minh 1
city 1
study 1
at 1
TDTU 1
```